Finding semantical differences between C++ standard versions

Tibor Brunner, Norbert Pataki, Zoltán Porkoláb

Programming languages are also participants of the modern software technology. Software systems are getting increasingly large-scale, which means that their code base also gets more and more complex. To compensate this, programming languages are providing high-level constructions to ease the handling of higher abstraction levels.

C++ is one of the most widespread programming languages among those applications which require high runtime performance. In the last few years C++ has introduced new features to enable creation of faster programs with less code. Among others C++11 handles multi-threading for parallel tasks, and move semantics for administration of temporary objects to save memory and CPU usage.

Programmers' expectation is that they can benefit from these new language features, and their previously written codes have the same behaviour when a new compiler version is used. Unfortunately this is not the case. We have found some examples when building the same program with a new compiler may have different results. This is caused by the semantic change of some constructions between two versions of the language standard. This means that it can be a challenging task to maintain the huge amount of legacy code, developed over the years, and to ensure that their behaviour is unchanged after an update of the build environment.

In this article we draw the attention on some of the semantic changes between the different versions of the standard. We present a static analysis method for finding these differences, and provide a tool for listing them. For this purpose we use LLVM/Clang compiler infrastructure.